Collectives for Multiple Resource Job Scheduling Across Heterogeneous Servers

Kagan Tumer
NASA Ames Research Center
Mailstop 269-4
Moffett Field, CA 94035
kagan@email.arc.nasa.gov

John Lawson
NASA Ames Research Center
Mailstop 269-2
Moffett Field, CA 94035
lawson@email.arc.nasa.gov

ABSTRACT

Efficient management of large-scale, distributed data storage and processing systems is a major challenge for many computational applications. Many of these systems are characterized by multi resource tasks processed across a heterogeneous network. Conventional approaches, such as load balancing, work well for centralized, single resource problems, but breakdown in the more general case. In addition, most approaches are often based on heuristics which do not directly attempt to optimize the world utility. In this paper, we propose an agent based control system using the theory of collectives. We configure the servers of our network with agents who make local job scheduling decisions. These decisions are based on local goals which are constructed to be aligned with the objective of optimizing the overall efficiency of the system. We demonstrate that agents configured using collectives outperform both team games and load balancing, by up to four times for the latter.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems. I.2.6 Learning

Keywords

Reinforcement learning, Job Scheduling, Computational Grid, Multi-resource optimization, Collectives

1. INTRODUCTION

With increasing demand for supercomputing resources (e.g., biological applications), the ability of a system to efficiently schedule and process jobs is becoming increasingly important. As such, heterogeneous computational grids where jobs can enter the network from any point and be processed at any point are becoming increasingly popular. For the single-resource case, this problem has been extensively studied [2]. However, multi-resource job scheduling across a network of heterogeneous servers (e.g., CPU speed, memory) is a difficult problem that has received much less attention [1].

Load balancing (LB) has been successfully applied to single resource scheduling problems. In its simplest form, load balancing aims at ensuring that the level of activity on each server stays the same, i.e., the load on the system is balanced across all the servers. Load balancing though *assumes* that the load being distributed across the servers is a de-facto desirable solution. In the multi-resource case, this assumption leads to suboptimal solutions [1]. The agent based approach we propose sidesteps this potential mismatch between balancing the load across the network and optimizing the world utility function. As long as that system behavior is good for the world utility, no consideration is being made to "split" the load or make the jobs processing "fair" in any way.

A traditional reinforcement learning, multi-agent systems for this problem would consist of an agent receiving either the full world reward (e.g., team game), or a reward concerning only its actions (e.g., a selfish reward). In general, team game solutions suffer from the signal-to-noise problem in which an agent has a difficult time discerning the effects on its actions on its utility, because that "signal" is getting swamped by the "noise" of the other agents. Selfish utilities on the other hand suffer from coordination issues, where actions that may be beneficial to one agent may cause significant damage to the system overall.

The theory of collectives [4] is concerned with overcoming the shortcomings of team games and selfish utilities. [1] In particular, it is concerned with providing agents with with rewards that are both "learnable" i.e., they have good signal-to-noise ratios, and are "factored" i.e., the utilities are aligned with the world utility.

2. SYSTEM MODEL

We modeled such a computational system as a network of N servers each with K resources $(r_1, ...r_k)$. Each server had a specified capacity for each resource assigned to be an integer ranging from $[1, M]$. Thus, M was a measure of the heterogeneity of the resources. Jobs were also specified by K resource requirements ranging from $[1, M]$. Each server had its own wait queue for jobs. Jobs entered the local queues either externally or were shipped from other servers.

If the processor was available, and the resource requirements met, the server would activated the first job in the queue. If the processor was available, but the server did not have the resource capacity to run the job, the server would remain idle until the problem job was sent to another server. This is expected to be one the main causes of bottlenecks

[1] A collective is defined as a multi agent system in which there is a well-defined world utility function that needs to be optimized, and where each agent takes actions based on its own private utility [3].

AAMAS'03, July 14–18, 2003, Melbourne, Australia.
ACM 1-58113-683-8/03/0007.

in the system and will be an issue that an intelligent job management system will need to address.

For a K resource-problem, we assigned 2^K agents per server and partitioned the space of jobs each agent has to deal with. Each agent had a a vector $\vec{p}$ whose components give the probability of routing a job to its neighbors. So the agents' actions are to set their own probability vector.

At each time step τ, new jobs were added to the system and placed in the wait queue of randomly selected servers. In particular, each server had a probability r of receiving a new job at each time. If a given processor was idle, and the first job in the queue met the resource requirements, that job would be activated. If not, the server would remain idle. In addition, for each τ, the server would make a decision about the first job in the queue, deciding whether to keep the job or sent it to a neighboring server. These decisions were made based on the agents' probability vectors which in turn are set using reinforcement learning algorithms.

Thus, there were two main sources of inefficiency in the system. The first were the bottlenecks created by jobs whose requirements exceeded the capacity of their server. When such a job got to the front of the queue, the server remained idle until the job was shipped to a neighbor. The second source of inefficiency arose from mismatches between a processor's speed and a job's cycle requirement.

We distinguish between two time scales: τ gives the time steps at which the jobs enter the system, move between queues, and are processed, whereas t gives the time steps at which the agents observe their utilities, change their actions, etc. This distinction is important because it is the only way an agent can get a "signal" from the system that will reflect the impact of its decision, i.e, the system has to settle down before a reward can be matched to an action. Therefore, an agent η changes its probability vector at each time t. Within a single time step t though, many jobs enter the system, are executed, routed etc. each of which occurs at time interval τ ($t >> \tau$).

For this problem, the world utility, G, is the weighted ratio of the all the jobs that were processed at time step t to all jobs that entered the system at that time step. The "difference" utility (DU) of an agent is the weighted fraction of jobs that were touched by that agent to the jobs that entered the system. This is different than a "selfish" utility (SU) which is the ratio of the jobs processed by the system at time step t to the total jobs that passed through that agent. (Note DU concerns the agent's impact on the system whereas SU only concerns the agents' success at getting "its own jobs" processed.)

3. RESULTS

We ran extensive simulations on networks of $N = 50$ servers having $K = 2$ resources, and compared our agent-based approach against a fixed, deterministic version of multi-resource load balancing. The 50 servers had 4 agents each, making for 200 total agents. (The results were averaged over 50 different randomly generated network configurations.) The servers were connected into a network having a ring configuration with random connections added in the spirit of "small world's" networks. In general, each server had 2-4 neighbors with which it had a direct connection.

Tables 1-2 show results for the absolute and relative performance[2] for the different algorithms at $t = 400$. Notice

[2]Because it wasn't always possible to attain 100% efficiency we provide an estimate for the upper bounds on performance based on the number of jobs in the system. The percentage *gain* reflects how much of the gap between random and the upper bound is covered by the algorithm.

Table 1: System Processing Efficiency (r=0.2,M=8)

Algorithm	Net Efficiency	Perc Gain
Opt Estimate	1.0	-
RAND	0.6435	-
SU	0.6345	-2.53%
TG	0.6703	7.51%
DU	0.7932	41.97%
LB	0.2254	-117.28%

Table 2: System Processing Efficiency (r=0.8,M=2)

Algorithm	Net Efficiency	Perc Gain
Opt Estimate	0.781	-
RAND	0.6260	-
SU	0.6140	-7.78%
TG	0.6376	7.48%
DU	0.6911	41.98%
LB	0.6446	11.97%

that in all cases the learning based approaches are competitive or significantly outperform load balancing. Load balancing performs poorly for high M (high heterogeneity). In fact, even setting the probability vectors at random (RAND) outperforms load balancing for $M = 8$, $r = 2$. It is also in these large M regimes that approaches based on adaptive learning algorithms would be expected to do well. Simulations results show large increases in performance by having the probability vectors set using reinforcement learning.

These results also show the importance of setting the agents' personal utilities to be functions that are both "factored" and "learnable". The team game (TG) utility is factored trivially, but has poor learning properties for the individual agents since it includes information from the full system. The selfish (SU) utility is expected to be more learnable since it only includes effects of individual agents, but it is not factored (aligned with the world utility), and therefore could be doing a good job of learning the wrong thing. The difference utility (DU) derived using the theory of collectives is both factored and learnable. It consistently outperforms TG and SU for all parameter pairs (r, M).

4. REFERENCES

[1] W. Leinberger, G. Karypis, V. Kumar, and R. Biswas. Load balancing across near-homogeneous multi-resource servers. In *Proc. 9th Heterogeneous Computing Wksp*, pages 61–70, Cancun, Mexico, 2000.

[2] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Comp. Soc. Press, 1995.

[3] K. Tumer, A. Agogino, and D. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proc. of the First International Joint Conf. on AAMAS*, pages 378–385, Bologna, Italy, July 2002.

[4] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.